Features

- **Utilizes the AVR® Enhanced RISC Architecture**
  - **High Performance and Low Power**
  - **Sleep Mode to Conserve Power**
- **120 Powerful Instructions - Most Single Clock Cycle Execution**
- **32 x 8 General Purpose Working Registers**
- **Operating Range: 1.6 to 3.6 Volts**
- **Fully Static Operation, 0-33 MHz (0.5 micron), 0-45 MHz (0.35 micron)**
- **Seven External Interrupt Sources**
- **AVR Scalable Test Access Interface**
- **Test Vectors for >99% Fault Coverage**
- **Verilog and VHDL Simulation Models**
- **Faster Version can be Created Upon Request**

Description

The AVR® Embedded RISC Microcontroller Core is a low-power CMOS 8-bit microprocessor based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, it achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR Core is based on an enhanced RISC architecture that combines a rich instruction set with the 32 general purpose working registers. Each of the 32 registers is directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The architecture supports high level languages efficiently as well as extremely dense assembler code programs. It also provides any number of external and internal interrupts.

The AVR Core is provided in an encrypted netlist format with Verilog and VHDL simulation models, a fully functional test bench and ATPG vectors for >99% fault coverage. It is supported with a full suite of program and system development tools including: macro assemblers, ANSI C Compilers, program debugger/simulators, and in-circuit emulator.

Embedded RISC Microcontroller Core

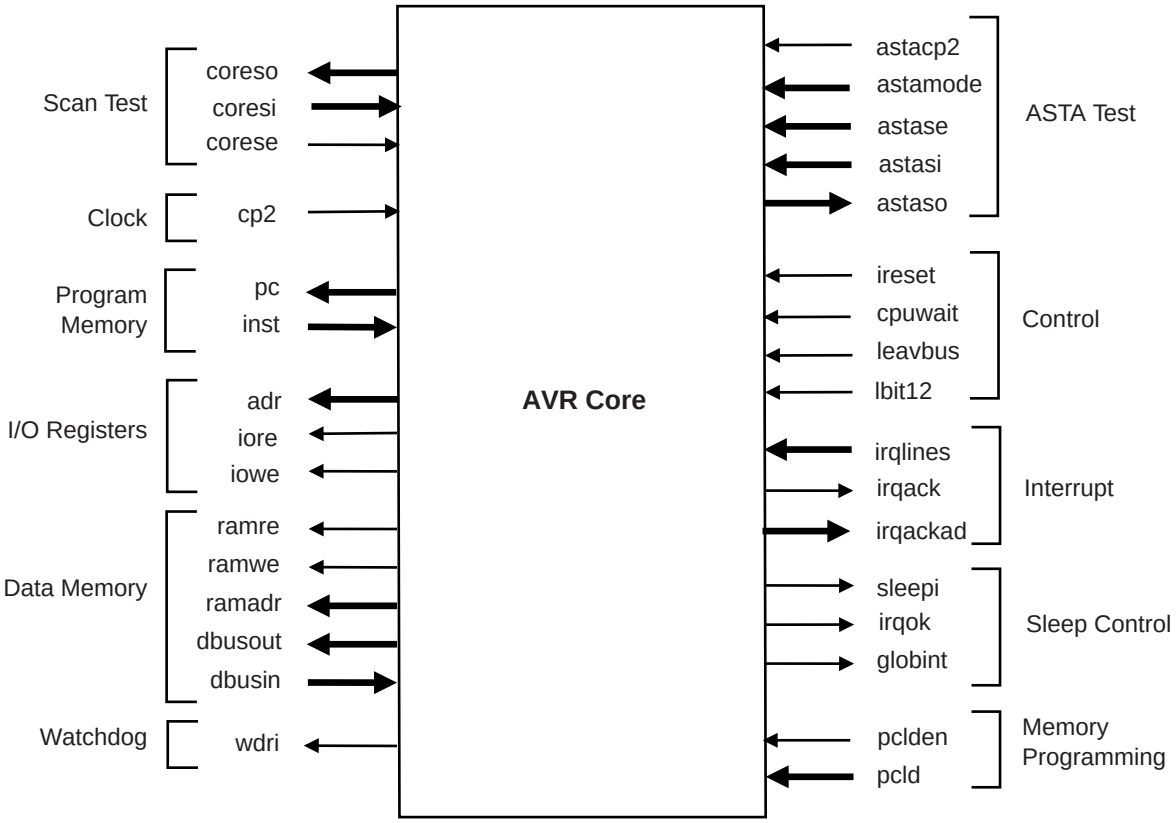
AVR®

Summary

Rev. 0890BS–05/99

I/O Configuration

Figure 1. AVR Core I/O Configuration



I/O Description

Table 1. I/O Description

Name	Input/ Output	Function
Clock Port		
cp2 Clock	Input	Any register in the core will update its contents only on the positive edge of **cp2**.
Control Ports		
ireset *AVR* Core Reset	Input	When high, **ireset** causes the core to reset the program counter **pc**, the status register SREG, and the stack pointer, loading all with zeros (H0000). When **ireset** is high and **leavbus** is inactive, zero (H00) is driven on the Data Bus **dbusout**, and the I/O Write Strobe **iowe** is held high while the I/O Read Strobe and the Data Memory Strobes (**ramre**, **ramwe**) are held low. This allows I/O registers to be reset by reading zero from the Bus.
cpuwait Wait CPU	Input	This signal is used to add wait cycles to allow slow memory accesses. When **cpuwait** is high, the core repeats the current cycle (only for instructions addressing the RAM space such as “ld” or “st”). When **cpuwait** is released, the cycle is executed as normal. For details, refer to the timing diagrams below.

Table 1. I/O Description (Continued)

Name	Input/ Output	Function
leavbus leave dbusout	Input	This signal is used to control **dbusout** externally. When high, **dbusin** is connected directly to **dbusout**, and all I/O and Data Memory Strobes are held low.
lbit12 Logical and between Lock bit 1 and 2	Input	Disables “lpm” and “elpm” instructions.
Program Memory Ports		
pc [15:0] Program Counter	Output	Program Memory always returns the instruction stored at the address pointed to. The size of this port determines the program memory size.
inst [15:0] Program Memory data bus	Input	Instruction from Program Memory is presented to the core, selected by the address on **pc** address bus.
I/O Registers		
adr [5:0] I/O Register address bus	Output	Valid only when accompanied by a strobe on **iore** or **iowe** lines.
iore I/O Registers read strobe	Output	Used only with the 64 I/O memory locations. These locations can be mapped into the regular Data Memory Address Space. The core will then issue an **iore** or **ramre** read strobe based on target address.
iowe I/O Registers write strobe	Output	Used only with the 64 I/O memory locations. These locations can be mapped into the regular Data Memory Address Space. The core will then issue an **iowe** or **ramwe** read strobe based on target address.
Data Memory Ports		
ramadr [15:0] Data Memory address bus	Output	Valid only when accompanied by a strobe on **ramre** or **ramwe** lines.
ramre Data Memory read strobe	Output	Used to address the SRAM memory locations. The core will issue an **iore** or **ramre** read strobe based on target address.
ramwe Data Memory write strobe	Output	Used to address the SRAM memory locations. The core will issue an **iore** or **ramre** read strobe based on target address.
dbusin [7:0] Data Bus Input	Input	All data transfers use **dbusin** or **dbusout** to transfer data into or out of the core. Memory locations are selected by the address on **ramadr** (Data Memory Address). I/O Register locations are selected by the address on **adr**.
dbusout [7:0] Data Bus Output	Output	
Interrupt Ports		
irqlines [6:0] Interrupt Request Lines	Input	Each interrupt source drives its own dedicated IRQ line into the Core. When the global interrupt bit is enabled, a high level (one) on any interrupt line will push the current **pc** on the stack. The associated interrupt handler vector address is put in the Program Counter **pc** before execution is restarted.
irqack Interrupt Acknowledge	Output	**irqack** will go high (one) for one clock cycle to acknowledge the interrupt being executed. This is often used as input to interrupt flags designed to clear when their corresponding interrupt handler is executed. The **irqackad** lines identify which interrupt is being executed during the same cycle.

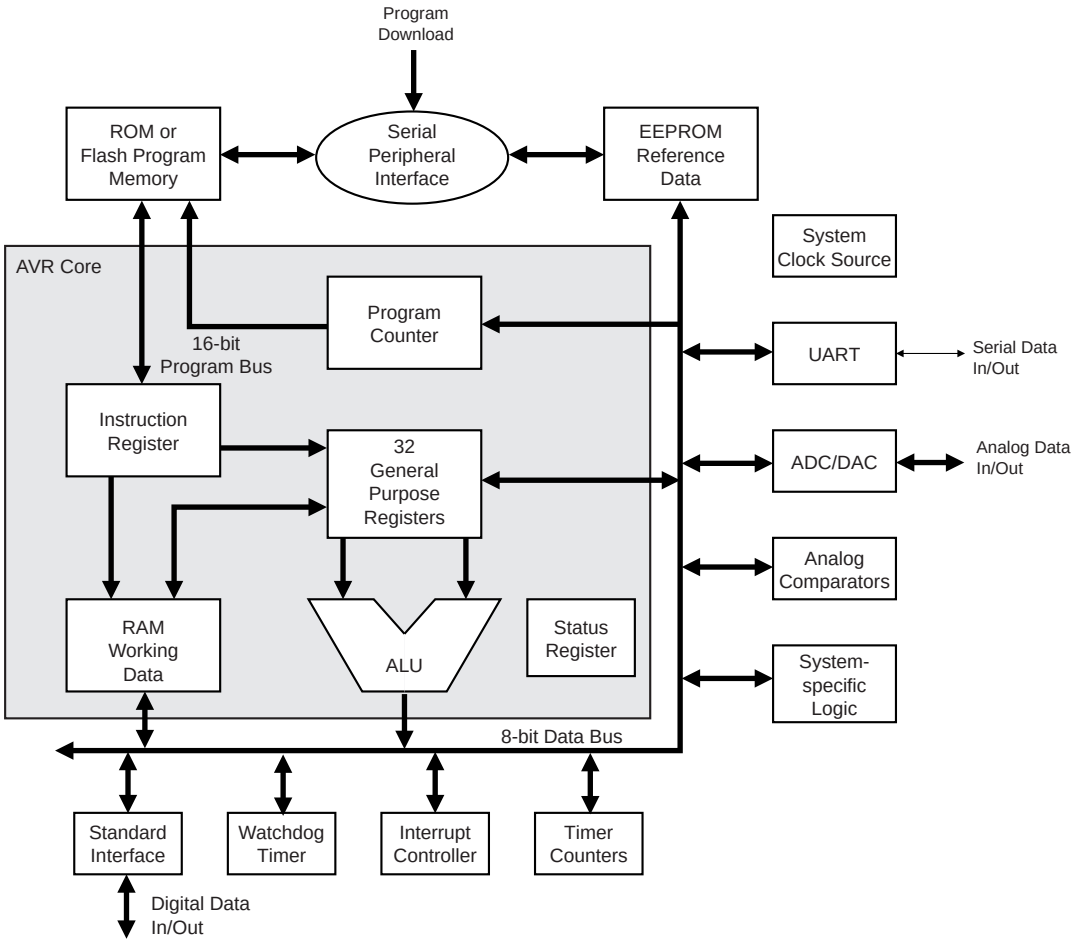
Table 1. I/O Description (Continued)

Name	Input/ Output	Function
irqackad [2:0] Interrupt Acknowledge Address	Output	The address of the interrupt being executed. The address is valid only if the **irqack** signal is set (one).
Sleep Controller Ports		
sleepi Sleep instruction	Output	Set while executing the 'sleep' instruction. This should cause the sleep controller to stop the clock to the core if sleep mode has been enabled.
irqok Interrupt Request OK	Output	When in sleep mode (clock stopped), this signal will tell the sleep controller that an interrupt exists which should cause the clock to restart. The sleep controller should start the core clock as soon as possible.
globint Global interrupts enabled	Output	This is the current state of the I bit in the Core State Register. This signal is used to qualify wake-up from power-down by external interrupts.
Memory Programming Ports		
pclden enable **pc** load	Input	Enable **pc** load with **pcld** signals.
pcld [1:0] Load Program Counter	Input	Load Program Counter **pc** from **dbusin** if **pclden** is active. **pcld [1]** load high byte, **pcld [0]** load low byte.
Watchdog Port		
wdri Watchdog reset instruction	Output	Set while executing the 'wdi' (watchdog reset) instruction.
Scan Test Ports		
corese	Input	Core Test Scan Enable
coresi [2:0](1)	Input	Core Test Scan Inputs
coreso [2:0](1)	Output	Core Test Scan Outputs
ASTA Test Ports		
astacp2 ASTA clock	Input	Any register in the ASTA interface will update its contents only on the positive edge of **astacp2**.
astamode [1:0](1)	Input	ASTA mode inputs used to swap between ASTA mode or normal function mode.
astase [8:0](1)	Input	ASTA Test Scan Enables
astasi [8:0](1)	Input	ASTA Test Scan Inputs
astaso [8:0](1)	Output	ASTA Test Scan Outputs

Note: 1. Width is subject to change.

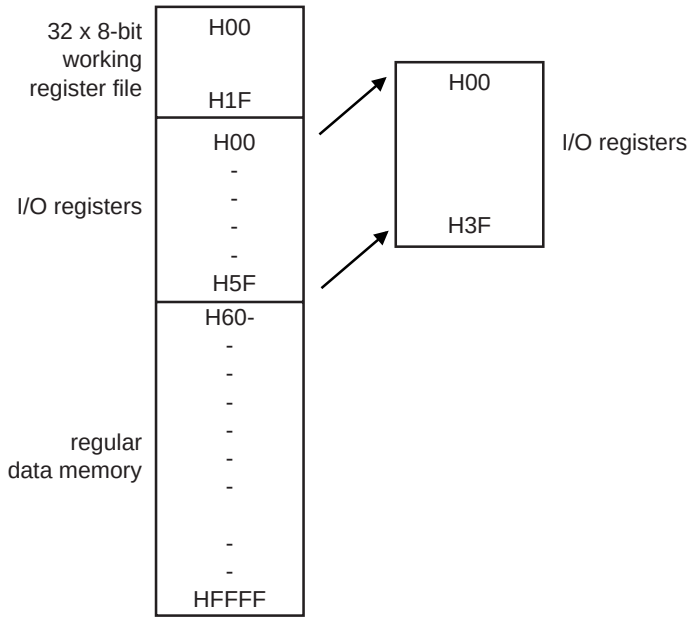
AVR Core Architecture

Figure 2. Block Diagram of the AVR Core and a Typical Set of Peripherals



The AVR core is based on a Harvard architecture with separate memories and buses for program and data (Figure 2). The memory spaces in the AVR architecture are all linear and regular memory maps.

Figure 3. AVR Data Memory Map



The central AVR architectural element is a fast-access register file containing 32 x 8-bit general purpose registers with a single clock cycle access time. This means that during one clock cycle, one ALU operation is executed. Two operands are accessed from the register file, the operation is executed, and the result is stored back in the register file - in one clock cycle. The ALU supports arithmetic and logic functions between registers or between a constant and a register, as well as single register operations.

The program memory can be implemented in ROM or Flash memory. It is accessed with a single level of pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This enables instructions to be executed in every clock cycle. All AVR instructions have a single 16-bit word format, meaning that every program memory address contains a single instruction. During interrupts and subroutine calls, the return address is stored on a software stack.

The 8-bit data memory (Figure 3) has 16-bit direct addressing. This gives a potential memory space of 64K bytes. The data memory address space includes the register file, and a 64-address I/O memory space for peripheral functions such as control registers, timer-counters and A/D converters. As shown in Figure 3, the I/O memory space is automatically re-mapped for access by the register file.

The General Purpose Register File

Although the working register file of the AVR is normally perceived as being a 32 by 8-bit storage unit, the register file is actually a 16 by 16-bit memory unit. The 16-bit wide data format is necessary to allow the core to update the 16-bit memory pointers in a single cycle.

Figure 4. Mapping between 32 x 8-bit registers and 16 x 16-bit array addresses

	15 high 8	7 low 0
Address: H00	R1	R0
H01	R3	R2
H02	R5	R4
H03	R7	R6
H04	R9	R8
H05	R11	R10
H06	R13	R12
H07	R15	R14
H08	R17	R16
H09	R19	R18
H0A	R21	R20
H0B	R23	R22
H0C	R25	R24
H0D	R27	R26
H0E	R29	R28
H0F	R31	R30

Registers R31 - R30 (the Z-register), R29 - R28 (the Y-register) and R27 - R26 (X-register) can be used for indirect addressing.

Arithmetic Logic Unit

The high-performance AVR Arithmetic Logic Unit (ALU) operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, ALU operations between registers are executed. ALU operations are divided into three mail categories: arithmetic, logic and bit-functions. It is possible to implement a hardware multiplier in the arithmetic part of the ALU.

Program and Data Addressing Modes

The AVR core supports powerful and efficient addressing modes for program instructions and data. These include register direct (one or two registers), register indirect (via the X, Y or Z-register), I/O direct and relative program addressing (Program Counter plus offset in the Instruction Register).

I/O Registers

All the peripheral status, control and data registers are grouped into a 64-byte I/O space. This can be accessed directly from the register file, and from within the I/O space of the data memory. See Figure 3. The required re-mapping between the two address spaces is performed automatically by the processor.

Status Register

The Status Register (SREG) is updated after all arithmetic and logical instructions, as well as by dedicated instructions. Software can also access this register to store or manipulate its contents. It contains a set of status flags, as follows:

- I: Global Interrupt Enable
- H: Half Carry Flag
- V: Twos Complement Overflow Flag
- Z: Zero Flag
- T: Bit Copy Storage
- S: Sign Bit
- N: Negative Flag
- C: Carry Flag

The Stack Pointer

The 16-bit Stack Pointer (SP) is built up from two 8-bit registers in the I/O space. It points to the area in data memory where the subroutine and interrupt stacks are located. This stack space must be defined by the program before any subroutine calls are executed or interrupts are enabled.

Interrupt Handling

A prioritized interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the status register. Each interrupt has a separate interrupt vector in the interrupt vector table at the beginning of the program memory. They are prioritized by address: the lower the interrupt vector address, the higher the priority.

AVR Scalable Test Access (ASTA) Interface

The AVR Scalable Test Access (ASTA) interface provides designers with great flexibility to test the AVR Embedded Core and its peripherals. First, the ASTA architecture allows the designer to apply pre-computed ATPG test vectors with more than 99% fault coverage. Secondly, it allows ATPG vectors to be generated for the rest of the chip. The main characteristic of the ASTA architecture however, is its capability to be scaled and split into several scan chains, making it possible to test the program memory space, the RAM space and the I/O space simultaneously.

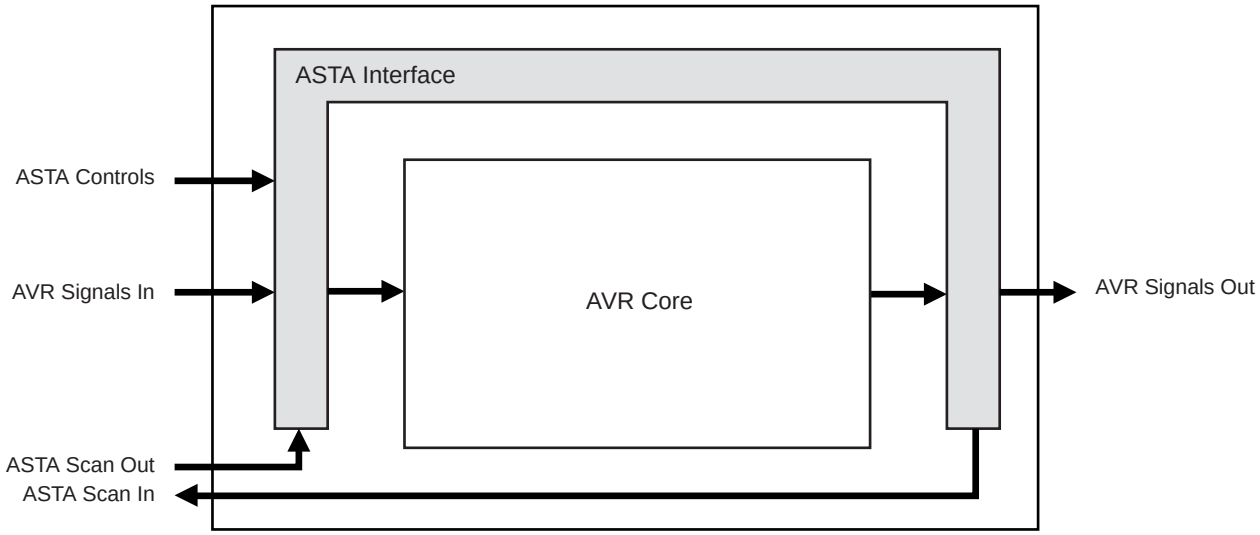
The ASTA interface can be considered as a boundary scan ring that encompasses the entire AVR Embedded Core. This scan chain allows all primary AVR inputs to be controlled and all primary AVR outputs to be observed, resulting in over a 99% fault coverage. This scan ring is actually split into nine different scan chains which can be grouped as desired, giving the flexibility to create specific tests such as RAM space testing or program memory space testing.

The AVR Embedded Core is shipped with a pre-computed set of ATPG test vectors ensuring over a 99% fault coverage. This set of vectors is generated with a special configuration of both the ASTA interface and scan signals. The designer must recreate this configuration in his design.

In order to apply the precomputed ATPG vectors, the designer must have access to the following top level pins:

- clock (cp2)
- ASTA test interface
  (special configuration requires only 6 signals)
- scan test signals

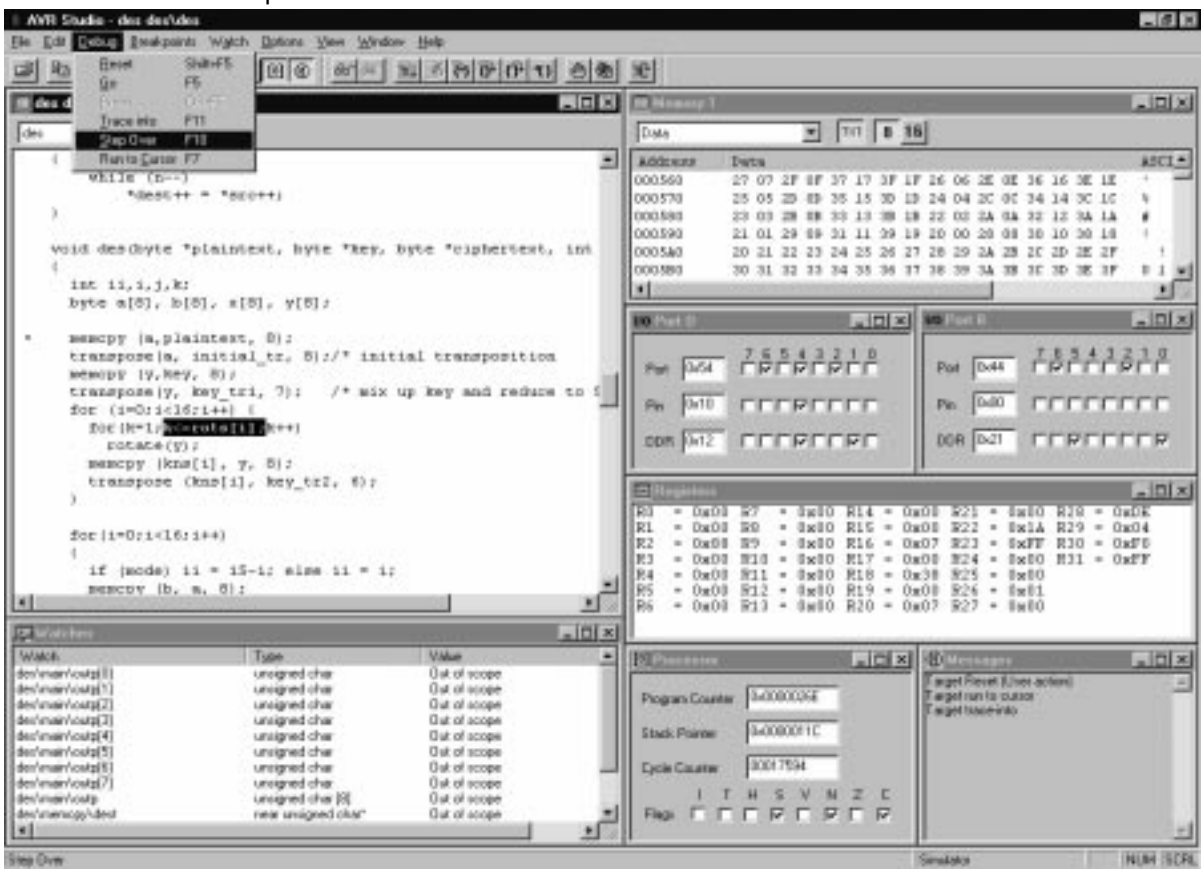
Figure 5. The ASTA Interface



Software Development Toolset

The AVR core is delivered with a comprehensive set of development tools for rapid creation and updating of applications. These include: ANSI-compliant C compilers, macroassemblers, linkers, debuggers, simulators, in-circuit emulators, and evaluation boards.

Figure 6. AVR Software Development Tools



AVR Assembler

The AVR Assembler translates the Assembler source programs into object code. It is extremely fast and can automatically jump to the next and/or previous error. The Assembler has powerful macro capabilities and supports all standard output formats. It has an easy to use MS-Windows interface but an MS-DOS® command line version is also available. An editor is included in the MS-Windows version.

AVR Simulator

The AVR Simulator has powerful debugging facilities. Simulation is realized by an assembly source level. The simulator is fully supported by AVR peripheral devices and is easy to use in the MS-Windows interface.

IAR Development Tools for Atmel AVR

The IAR Development Tools contain a fully ANSI Compatible C Compiler and include an Embedded Workbench. They function under DOS, Windows® 3.11, Windows 95® and Windows NT®.

Atmel AVR ASIC In-Circuit Emulator

The ASIC In-Circuit Emulator (ASIC-ICE) gives a full visibility of all AVR core resources. It has powerful breakpoint facilities and an extensive execution control.

The AVR ASIC-ICE is based on the established ICEPRO technology and is completed with an FPGA-based POD (the external card between the ICE and the application) for flexibility. It should be noted that the ASIC-ICE is a core emulator and usually requires that the rest of the ASIC is implemented on a custom POD (FPGA or ASIC based). On-chip ASIC-ICE emulation within a single ASIC is only possible with a POD-compatible ASIC (i.e. bound-out core with 57 active POD interface signals).

Provided with a real-time emulation, a software adjustable clock speed and a serial and parallel port interfacing, the Atmel AVR ASIC In-Circuit Emulator is fully integrated with other AVR development tools.

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686-677
FAX (44) 1276-686-697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

0890BS–05/99/xM